Splicing Systems: Accepting Versus Generating

Juan Castellanos Victor Mitrana and Eugenio Santos

Abstract. In this paper we propose a condition for rejecting the input word by an accepting splicing system which is defined by a finite set of forbidding words. More precisely, the input word is accepted as soon as a permitting word is obtained provided that no forbidding word has been obtained so far, otherwise it is rejected. Note that in the new variant of accepting splicing system the input word can be rejected if either no permitting word is ever generated (like in [10]) or a forbidding word has been generated and no permitting word had been generated before. We investigate the computational power of the new variants of accepting splicing systems. We show that the new condition strictly increases the computational power of accepting splicing systems. Rather surprisingly, accepting splicing systems considered here can accept non-regular languages, a situation that has never occurred in the case of (extended) finite splicing systems without additional restrictions.

1 Introduction

One of the basic mechanism by which genetic material is merged is the recombination of DNA sequences under the effect of enzymatic activities. This process has been formalized as a word rewriting operation as follows: the restriction enzymes have been approximated by a finite set of rules defining the restriction sites and the DNA sequences, on which the enzymes act, have been approximated by a finite set of words usually called axioms. This is actually the main idea of the *splicing* operation viewed as a language theoretical approach of the recombinant behavior of DNA under the influence of restriction enzymes and ligases considered by T. Head in [7]. Roughly speaking, the splicing operation is applied to two DNA sequences (represented by words) which are cut at specific sites (represented by splicing rules), and the first subword of one sequence is pasted to the second segment of the other and vice versa. A new formal device to generate languages based on the iteration of splicing operation has been

* Work partially supported by the Alexander von Humboldt Foundation.

considered. Known as *splicing system*, this computation model has been vividly investigated in the last two decades. In spite of the vast literature devoted the topic, the real computational power of finite splicing systems is still partially unknown as the characterization of languages generated by these systems is an open problem. The problem is completely solved for extended splicing systems (a terminal alphabet is used for squeezing out the result), i.e. extended splicing systems are computationally equivalent to finite automata. Another large part of the research in this area has been focused on defining different types of splicing systems and investigating their computational power from a language generating point of view. Many variants of splicing systems have been defined and investigated; we mention here just a few of them: distributed splicing systems [3], splicing systems with multisets [5], splicing systems with permitting and forbidding contexts [6], programmed and evolving splicing systems [14]. Under certain circumstances, splicing systems are computationally complete and universal (see [15] for an overview). This result suggests the possibility to consider splicing systems as theoretical models of programmable universal DNA computers based on the splicing operation.

Several other works like [1], and the references therein, address two fundamental questions concerning splicing systems: *recognition*, which asks for an algorithm able to decide whether or not a given regular language is a splicing language, and *synthesis*, which asks for an effective procedure to construct a splicing system able to generate a given splicing language.

In [10] a novel look on splicing systems is proposed, namely splicing systems are viewed as language accepting devices and not generating ones. More precisely, a usual splicing system is used for accepting/rejecting an input word in accordance with some predefined accepting conditions. The new computational model was called *accepting splicing system*. It is rather strange that though the theory of splicing systems is mature and well developed, an accepting model based on the splicing operation has not considered so far with two exceptions:
– Work [9], where two well-known NP-complete problems were solved with a variant of accepting splicing systems with regular sets of splicing rules. This variant with finite sets of splicing rules was further investigated in [8].
– Work [2], where a splicing recognizer that computes by observing and contains a part exhibiting some similarity to the accepting splicing system defined in [10].
Two ways of iterating the splicing operation and two variants of accepting splicing system are investigated in [10]. Altogether, one obtains four models which are compared with each other as well as with the generating splicing systems from the computational power point of view.

This work is a continuation of [10]. While the accepting splicing systems considered in [10] reject the input word only if no word (considered as a permitting word) from a given finite set is obtained during the splicing process, in this paper we propose a similar condition for rejecting the input word. This condition is also defined by a finite set of words considered as forbidding words. More precisely, the input word is accepted as soon as a permitting word is obtained provided that no forbidding word has been obtained so far, otherwise it is rejected. Note

that in the new variant of accepting splicing system the input word can be rejected if either no permitting word is ever generated (like in [10]) or a forbidding word has been generated and no permitting word had been generated before. The main goal of this paper is to investigate the computational power of the new variants of splicing systems. Clearly, the new variants are at least as powerful as the variants considered in [10]. We actually show that the new condition strictly increases the computational power of accepting splicing systems. Rather surprisingly, accepting splicing systems considered here can accept non-regular languages, a situation that has never occurred in the case of (extended) finite splicing systems without additional restrictions.

2 Basic Definitions and Notation

We start by summarizing the notions used throughout the paper. For all undefined notions the reader may consult [17]. An *alphabet* is a finite and nonempty set of symbols. Any finite sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^*, the empty word is denoted by ε, and the length of the word x is denoted by $|x|$. If $w = xyz$ with x, y, z being non-empty words, then x is a prefix of w, z is a suffix of w, and y is a subword for w. Moreover, we write $x^{-1}w = yz$ and $wz^{-1} = xy$. By convention, if x is not a prefix (suffix) of y, then $x^{-1}y = y$ ($yx^{-1} = y$). For two sets of words A and B, we write $A^{-1}B = \{x^{-1}y \mid x \in A, y \in B\}$ and $AB^{-1} = \{xy^{-1} \mid x \in A, y \in B\}$. For a word x we denote by $\mathrm{Pref}_k(x)$, $\mathrm{Suff}_k(x)$ and $\mathrm{Inf}_k(x)$, the prefix, suffix and the set of subwords of x, respectively.

Note that we ignore the empty word when we define a language and the empty set when we define a class of languages.

A splicing rule over V is 4-tuple $[(u_1, u_2); (u_3, u_4)]$, with $u_1, u_2, u_3, u_4 \in V^*$. For a splicing rule $r = [(u_1, u_2); (u_3, u_4)]$ and a pair of words $x, y \in V^*$, we write

$$\begin{aligned}\sigma_r(x, y) = &\{y_1u_3u_2x_2 \mid x = x_1u_1u_2x_2, y = y_1u_3u_4y_2\} \\ &\cup \{x_1u_1u_4y_2 \mid x = x_1u_1u_2x_2, y = y_1u_3u_4y_2\}\end{aligned}$$

for some $x_1, x_2, y_1, y_2 \in V^*$. This definition is extended to a set of splicing rules R and a language L by

$$\sigma_R(L) = \bigcup_{r \in R} \bigcup_{w_1, w_2 \in L} \sigma_r(w_1, w_2).$$

Without risk of confusion, we also denote for two languages L_1, L_2

$$\sigma_R(L_1, L_2) = \bigcup_{x_1 \in L_1} \bigcup_{x_2 \in L_2} \sigma_R(x_1, x_2), \text{ where } \sigma_R(x_1, x_2) = \bigcup_{r \in R} (\sigma_r(x_1, x_2).$$

A *generating splicing system* (*GenSS* for short) is a construct

$$H = (V, A, R),$$

where V is an alphabet, $A \subseteq V^*$ is the initial language, and R is a set of splicing rules over V. For a splicing system $H = (V, A, R)$ we set

$$\begin{aligned} &\sigma_R^0(A) = A, \\ &\sigma_R^{i+1}(A) = \sigma_R^i(A) \cup \sigma_R(\sigma_R^i(A)), i \geq 0, \\ &\sigma_R^*(A) = \bigcup_{i \geq 0} \sigma_R^i(A). \end{aligned} \qquad (*)$$

When the set of splicing rules is clear, we omit the subscript. Then, the language generated by H is defined as $L(H) = \sigma_R^*(A)$. Adding a terminal alphabet T we get an *extended generating splicing system* $H = (V, T, A, R)$, $T \subseteq V$, which generates the language $L(H) = T^* \cap \sigma_R^*(A)$. As all systems considered in this paper are extended systems, we shall omit the word "extended". Given a generating splicing system H as above, we say that a word $w \in L(H)$ is a *proper word* of $L(H)$, if it is generated in at least one splicing step. Clearly each word in $L(H) \setminus A$ is proper. The class of languages generated by $GenSS$ is denoted by $\mathcal{L}(GenSS)$.

An important result in splicing theory is the so-called *Regularity Preserving Lemma* proved first in [4], as a consequence of a more general result, and then in [16] by a direct argument. It states that $GenSS$ with a finite set of rules and a finite initial language, i.e. A and R are both finite sets, generate exactly the class of regular languages [13]. When one allows the set of splicing rules (written as words like in [12]) to be described by regular expressions, we obtain computationally complete systems [12].

For a $GenSS$ $H = (V, T, A, R)$ we also introduce the following non-uniform variant of iterated splicing, where the splicing is only done with axioms. More precisely, in the non-uniform case splicing at any step occurs between a generated word in the previous step and an axiom, differently from the general case where splicing at any step occurs between any two words generated in the previous steps. We set

$$\begin{aligned} &\tau_R^0(A) = A, \\ &\tau_R^{i+1}(A) = \sigma_R(\tau_R^i(A), A), i \geq 0, \\ &\tau_R^*(A) = \bigcup_{i \geq 0} \tau_R^i(A). \end{aligned} \qquad (\diamond)$$

The language generated by H in the non-uniform way is defined as $L_n(H) = \tau_R^*(A) \cap T^*$. The class of languages generated by $GenSS$ in the non-uniform way is denoted by $\mathcal{L}_n(GenSS)$.

Theorem 1. [13,10] *Both $\mathcal{L}(GenSS)$ and $\mathcal{L}_n(GenSS)$ equal the class of regular languages.*

We now introduce the definitions and terminology for accepting splicing systems. An *accepting splicing system* ($AccSS$ for short) is a 6-tuple

$$\Gamma = (V, T, A, R, P, F),$$

where V is an alphabet, $H_\Gamma = (V, T, A, R)$ is a splicing system, while P and F are finite sets of words over V. The elements of P are called *permitting words* while those of F are called *forbidding words*.

Let $\Gamma = (V, T, A, R, P, F)$ be an *AccSS* and a word $w \in V^*$; we define the following iterated splicing that is slightly different from (*):

$$\sigma_R^0(A, w) = \{w\},$$
$$\sigma_R^{i+1}(A, w) = \sigma_R^i(A, w) \cup \sigma_R(\sigma_R^i(A, w) \cup A), i \geq 0,$$
$$\sigma_R^*(A, w) = \bigcup_{i \geq 0} \sigma_R^i(A, w).$$

Although this operation and that defined by (*) are denoted in the same way, there is no risk of confusion as that defined by (*) is an one-argument function while that defined here has two arguments. We say that the word $w \in T^*$ is accepted by Γ if there exists $k \geq 0$ such that

$$(i) \quad \sigma_R^k(A, w) \cap P \neq \emptyset,$$
$$(ii) \quad \sigma_R^k(A, w) \cap F = \emptyset.$$

The following short discussion is in order. The reason for this definition of $\sigma_R^*(A, w)$ is two fold: on the one hand, we maintain a certain uniformity in the definitions of the two ways of acceptance by *AccSS* (see below) and on the other hand, we forbid axioms to be considered as permitting or forbidding words unless they are obtained as proper words. This restriction avoids a "funny" situation in which an *AccSS* accepts either every word whenever an axiom is a final word, or no word whenever an axiom is a forbidding word.

Remark 1. *The following sequence of inclusions is immediate:*

$$(\sigma_R^*(A \cup \{w\}) \setminus A) \subseteq \sigma_R^*(A, w) \subseteq \sigma_R^*(A \cup \{w\}).$$

On the other hand, the next equality will be useful in the sequel.

$$\sigma_R^*(A, w) = \sigma_R^*(A \cup \{w\}) \setminus \{x \in A \mid x \neq w,$$
$$x \text{ is not a proper word of } \sigma_R^*(A \cup \{w\})\}.$$

The language accepted by an *AccSS* Γ is denoted by $L(\Gamma)$.

Remark 2. *Note that every AccSS $\Gamma = (V, T, A, R, P, F)$ with $F = \emptyset$ is actually an (extended) AccSS considered in [10]. This remark suggests to consider for an AccSS $\Gamma = (V, T, A, R, P, F)$, the language $L^\emptyset(\Gamma) = L(\Gamma')$, where $\Gamma' = (V, T, A, R, P, \emptyset)$.*

The class of languages accepted by *AccSS* and *AccSS* without forbidding words is denoted by $\mathcal{L}(AccSS)$ and $\mathcal{L}^\emptyset(AccSS)$, respectively.

For an accepting splicing system $\Gamma = (V, T, A, R, P, F)$ we also introduce the following non-uniform way of accepting words similar to the non-uniform way of generating a language by a *GenSS*. The computation of such a system is nondeterministic; moreover the working mode of such a system involves words originating from the input word and a finite amount of information given by the set of axioms.

For an $AccSS = (V, T, A, R, P, F)$ and a word $w \in V^*$ we define the following non-uniform variant of iterated splicing, where the splicing is only done with

axioms, similarly to ($\diamond$):

$$\tau_R^0(A,w) = \{w\},$$
$$\tau_R^{i+1}(A,w) = \tau_R^i(A,w) \cup \sigma_R(\tau_R^i(A,w), A), i \geq 0,$$
$$\tau_R^*(A,w) = \bigcup_{i \geq 0} \tau_R^i(A,w).$$

The language accepted by Γ in the non-uniform way is defined by:

$$L_n(\Gamma) = \{w \in T^* \mid \exists k \geq 0(\tau_R^k(A,w) \cap P \neq \emptyset) \ \& \ (\tau_R^k(A,w) \cap PF = \emptyset)\}.$$

The class of languages accepted by $AccSS$ and $AccSS$ without forbidding words in the non-uniform way is denoted by $\mathcal{L}_n(AccSS)$ and $\mathcal{L}_n^{\emptyset}(AccSS)$, respectively.

3 Computational Power

The inclusions $\mathcal{L}_n^{\emptyset}(AccSS) \subseteq \mathcal{L}_n(AccSS)$ and $\mathcal{L}^{\emptyset}(AccSS) \subseteq \mathcal{L}(AccSS)$ are immediate from definitions. Furthermore, by Theorem 2 in [10] $\mathcal{L}_n^{\emptyset}(AccSS) \subset \mathcal{L}^{\emptyset}(AccSS)$ holds. The proof of this theorem can be easily completed to a proof for the inclusion $\mathcal{L}_n(AccSS) \subseteq \mathcal{L}(AccSS)$. Based on these observations we now state:

Proposition 1. *$\mathcal{L}_n^{\emptyset}(AccSS) \subset \mathcal{L}_n(AccSS)$ and $\mathcal{L}^{\emptyset}(AccSS) \subset \mathcal{L}(AccSS)$.*

Proof. It suffices to provide a language in $\mathcal{L}_n(AccSS) \setminus \mathcal{L}^{\emptyset}(AccSS)$. This language, say L, is defined by the regular expression a^+b^+. It is clear that a word is in L if and only if the following conditions are satisfied:

(i) it does not contain the subword ba;

(ii) it contains the subword ab.

We now construct an $AccSS$ that accepts L in the non-uniform way. Let

$$\Gamma = (\{a,b,\#,\$\}, \{a,b\}, \{\#\#,\$\$\}, R, \{\$ab\$\}, \{\#ba\#\}),$$

where $R = \{[(ba,\varepsilon);(\#,\#)], [(\varepsilon,ba\#);(\#,\#)], [(ab,\varepsilon);(\$,\$)], [(\varepsilon,ab\$);(\$,\$)]\}$.

By this construction, it is easy to note that if the input word contains the subword ba, then the forbidding word $\#ba\#$ is obtained in the second splicing step. As no permitting word can be produced in the first splicing step, actually no matter the input word, we infer that all input words as above are rejected by Γ. Consequently, a necessary (but not sufficient) condition for an input word to be accepted by Γ is to not contain the subword ba. On the other hand, a similar reasoning lead to the conclusion that the permitting word $\$ab\$$ is also obtained in the second splicing step provided that the input word does contain the subword ab. By these considerations, after the second step, Γ either accepts its input, provided it contains ab, but not ba, and rejects otherwise.

On the other hand, following [10], for every $AccSS$ $\Gamma = (V,T,A,R,P,\emptyset)$, there exists an integer $k > 0$ such that if $w \in L(\Gamma)$, with $|w| \geq k$, then $wyw \in L(\Gamma)$ for any $y \in T^*$. In conclusion, $\{a^nb^m \mid n,m \geq 1\} \notin \mathcal{L}^{\emptyset}(AccSS)$ which concludes the proof. □

It is worth mentioning here the very simple and efficient way to define the language in the proof of the previous proposition by an accepting splicing system (after two splicing steps only) in comparison with a generating splicing system. Actually, the class $\mathcal{L}_n(AccSS)$ contains "almost" all regular languages. More precisely,

Proposition 2. *For every regular language $L \subseteq V^*$ and $£ \notin V$, the language $£L \in \mathcal{L}_n(AccSS)$.*

Proof. Let $A = (Q, V\delta, q_0, Q_f)$ be a deterministic finite automaton accepting the language L. We construct the following $AccSS$:

$$\Gamma = (V \cup Q \cup \{£, \$, \#\}, V \cup \{£\}, Q\{\#\} \cup \{\#\#, \$\$\}, R, Q_f, \{\#£\#\}),$$

where

$$R = \{[(X£, \varepsilon); (\#, \#)] \mid X \in V \cup \{£\}\} \cup \{[(\varepsilon, £\#); (\#, \#)], [(£, \varepsilon); (\$, \$)]\} \cup \{[(\$, a); (q_0, \#)] \mid a \in V\} \cup \{[(qa, \varepsilon); (\delta(q, a), \#)] \mid a \in V\}.$$

As in the proof of the previous result, after the first two consecutive steps, the forbidding word $\#£\#$ is obtained provided that the input word is of the form $x£y$ with $|x| > 0$. Therefore, all input words of this form are rejected by Γ.

Let us now analyze the computation of Γ on an input word of the form $£y$, $y \in V^*$. In the first two splicing steps, one obtains consecutively $\$y$ and then q_0y. Note that the other by-product words are $£\$$ and $\$\#$ that cannot be further spliced. From now on, a word qz, $q \in Q$, $z \in V^*$, is computed at some step if and only if $y = xz$ and $\delta(q_0, x) = q$. In conclusion, an input word $£y$ is accepted by Γ if and only if $y \in L(A)$.

The proof is complete as soon as we note that every input word $y \in V^*$ is "inert" with respect to Γ, in the sense that no splicing can be done. □

We now prove a result which is rather unexpected as this situation has never occurred so far in the case of finite splicing systems, namely finite splicing systems able to define non-regular languages.

Proposition 3. *The non-regular language $\{a^nb^n \mid n \geq 1\}$ lies in $\mathcal{L}(AccSS)$.*

Proof. Let $\Gamma = (V, \{a, b\}, A, R, P, F)$ be the $AccSS$ defined by

$$V = \{a, b, \#, \$, ¢, £, ¥\},$$
$$A = \{\#\#, \$\$, ££, ¢¢, ¢¥, ¥¢\},$$
$$P = \{a¢¢b\}, \text{ and } F = \{\$ba\#, a¥¥, ¥¥b\},$$

and R contains the following rules which are accompanied by their role:

(i) $\{[(ba, \varepsilon); (\#, \#)], [(\varepsilon, ba\#); (\$, \$)]$. In two consecutive splicing steps the forbidding word $\$ba\#$ is generated, provided that the input word contains the subword ba.

(ii) $\{[(a,b);(£,£)],[(a,£);(¢,¢)],[(£,b);(¢,¢)]\}$. If the input word contains the subword ab, it is split into two parts $xa£$ and $£by$, with $x,y \in \{a,b\}^*$ in the first splicing step. In the next splicing step, the symbol $£$ is replaced by ¢ in both parts mentioned above. Note that in the first two splicing steps, no permitting word can be obtained. Therefore, every input word containing the subword ba is rejected by Γ. In conclusion, we analyze the computation of Γ on an input word of the form a^nb^m after getting the two words $a^n¢$ and $¢b^m$.

(iii) $\{[(\varepsilon,a¢);(¢,¢)],[(¢b,\varepsilon);(¢,¢)]\}$. The number of occurrences of a and b in the two words mentioned above is decreased simultaneously. Note that the end and beginning marker, respectively, remains unchanged, namely ¢.

(iv) $\{[(\varepsilon,a¢);(¢,¥)],[(¢b,\varepsilon);(¥,¢)]\}$. This marker can be changed to ¥.

(v) $\{[(a¢,\varepsilon);(\varepsilon,¢b)],[(¥,\varepsilon),(\varepsilon,¥)]\}$. With these rules, Γ comes to taking a decision. If $n=m$, then both words $a¢$ and $¢b$ have been eventually generated and the permitting word $a¢¢b$ is finally obtained. Let us analyze the splicing step when the forbidding word $a¥¥$ is obtained. This means that in the previous splicing step, both word $a¥$ and ¥ were obtained for the first time in the computation of Γ on the input word a^nb^m. Consequently, $n > m$ holds. It is worth noting that $a¢$ and $¢b$ are also available for splicing, so that the permitting word $a¢¢b$ and the forbidding word $a¥¥$ are obtained in the same splicing step. The case when the forbidding word $¥¥b$ is treated analogously.

In conclusion, the permitting word $a¢¢b$ is obtained before any forbidding word is obtained if and only if the input word is of the form a^nb^m with $n=m$. □

As it can be easily proved that the regular language $\{a^{2n} \mid n \geq 1\}$ does not belong to $\mathcal{L}(AccSS)$, we have:

Corollary 1. *The class of regular languages is incomparable with* $\mathcal{L}(AccSS)$.

We now consider an important subclass of regular languages that can be accepted by accepting splicing For a given $k>0$, and an alphabet V, we consider a triple $S_k = (A,B,C)$, where A, B and C are sets of words over V of length k. A language L over V is called *k-locally testable in the strict sense* (k-LTSS for short) if there exists a triple $S_k = (A,B,C)$ over V as above such that for any $w \in V^*$ with $|w| \geq k$, $w \in L$ iff $[\mathrm{Pref}_k(w) \in A, \mathrm{Suff}_k(w) \in B, \mathrm{Inf}_k(w) \subseteq C]$ ([11]). When L is specified by $S_k = (A,B,C)$, we write $L = L(S_k)$. A language L is called locally testable in the strict sense (LTSS) iff L is k-LTSS for some $k>0$. Clearly, every k-LTSS language is regular. A k-LTSS language L over V is *prefix-disjoint* if there exists a triple $S_k = (A,B,C)$ such that $L = L(S_k)$ and $(V^{-1}L) \cap (C \cup B) = \emptyset$. A *suffix-disjoint* k-LTSS language is defined analogously.

Proposition 4. *Every prefix-disjoint or suffix-disjoint k-LTSS language belongs to $\mathcal{L}_n(AccSS)$ for any $k \geq 1$.*

Proof. We assume that $L = L(A,B,C)$ is a prefix-disjoint k-LTSS language over the alphabet V and $S_k = (A,B,C)$ satisfies the prefix-disjoint condition. We construct the *AccSS* $\Gamma = (U,V,I,R,P,F)$, where

- $U = V \cup \{\langle x\rangle \mid x \in A \cup B \cup C\} \cup \{¢, \$\}$,
- $R = \{[(x, \varepsilon); (\langle x\rangle, ¢)] \mid x \in A\} \cup \{[(\varepsilon, ax¢); (¢, ¢)] \mid a \in V, x \in A\} \cup \{[(\langle x\rangle a, \varepsilon); (\langle y\rangle, \$)] \mid a \in V, y \in C \cup B, xa = by \text{ for some } b \in V\}$,
- $A = \{\langle x\rangle ¢ \mid x \in A\} \cup \{\langle x\rangle \$ \mid x \in C \cup B\}$,
- $P = \{\langle x\rangle \mid x \in B\}$ and $F = \{¢ax¢ \mid a \in V, x \in A\}$.

The working mode of Γ can be easily understood as soon as one makes an analogy with the construction in the proof of Proposition 2, where the words in A play the role of the marker $£$, and the symbols $\langle x\rangle$ play the role of the states. □

4 Final Remarks

The results proposed here are intended to improve the picture concerning the computational power of the accepting models based on the splicing operation as a counterpart of the well investigated generating splicing systems. However, there is still room for improving the overall picture. For instance, the precise relationship between the class of regular languages and each of the classes $\mathcal{L}_n^{\emptyset}(AccSS)$, $\mathcal{L}^{\emptyset}(AccSS)$ and $\mathcal{L}_n(AccSS)$ is

Another area of interest concerns the decidability properties of accepting splicing systems. The next result is just a beginning.

Theorem 2. *The membership problem is decidable for $\mathcal{L}(AccSS)$.*

Proof. Algorithm 1 solves the membership problem for an arbitrary $AccSS$ $\Gamma = (V, T, A, R, P, F)$: Note that the condition in line 1 is algorithmically testable as

Algorithm 1 Membership algorithm. **Input:** $w \in T^*$, $|w| = n$, $w \notin P$

```
1: if w ∉ L^∅(Γ) then
2:    return false; halt;
3: else
4:    for all k ≥ 1 do
5:       Q := σ_R^k(A, w);
6:       if (Q ∩ F ≠ ∅) then
7:          return false; halt;
8:       else
9:          if (Q ∩ P ≠ ∅) then
10:            return true; halt;
11:         end if
12:      end if
13:   end for
14: end if
```

the membership problem for $\mathcal{L}^{\emptyset}(AccSS)$ is decidable (see [10]). Moreover, if the condition from line 1 is satisfied, then the algorithm eventually halts within the cycle **for**. □

By [10], the emptiness and finiteness problems are decidable for $\mathcal{L}_n^{\emptyset}(AccSS)$. The status of these problems as well as of other decision problems for the accepting splicing systems considered here is still open.

Another investigation of interest in our view is to consider the accepting splicing systems introduced here as problem solvers like in [9]. To this aim, the property of an accepting splicing systems to make a decision after a finite number of splicing steps appears to be important. In other words, the rejection of the input word is always a consequence of reaching a forbidding word. None of the constructions proposed here has this property. Can each accepting splicing system be equivalently transformed into an accepting splicing system having this property?

References

1. Bonizzoni, P., Mauri, G.: Regular splicing languages and subclasses. Theoret. Comput. Sci. 340, 349–363 (2005)
2. Cavaliere, M., Jonoska, N., Leupold, P.: DNA splicing: computing by observing. Natural Computing 8, 157–170 (2009)
3. Csuhaj-Varjú, E., Kari, L., Păun, G.: Test tube distributed systems based on splicing. Computers and AI 15, 211–232 (1996)
4. Culik II, K., Harju, T.: Splicing semigroups of dominoes and DNA. Discrete Appl. Math. 31, 261–277 (1991)
5. Denninghoff, K.L., Gatterdam, R.W.: On the undecidability of splicing systems. Intern. J. Computer Math. 27, 133–145 (1989)
6. Freund, R., Kari, L., Păun, G.: DNA computing based on splicing. The existence of universal computers. Theory of Computing Syst. 32, 69–112 (1999)
7. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. Bull. Math. Biology 49, 737–759 (1987)
8. Loos, R., Malcher, A., Wotschke, D.: Descriptional complexity of splicing systems. Intern. J. Found. Comp. Sci. 19, 813–826 (2008)
9. Loos, R., Martín-Vide, C., Mitrana, V.: Solving SAT and HPP with accepting splicing systems. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 771–777. Springer, Heidelberg (2006)
10. Mitrana, V., Petre, I., Rogojin, V.: Accepting splicing systems. Theor. Comput. Sci. 411, 2414–2422 (2010)
11. McNaughton, R., Papert, S.: Counter-free automata. MIT Press, Cambridge (1971)
12. Păun, G.: Regular extended H systems are computationally universal. J. Automata, Languages, Combinatorics 1, 27–36 (1996)
13. Păun, G., Rozenberg, G., Salomaa, A.: Computing by splicing. Theoret. Comput. Sci. 168, 321–336 (1996)
14. Păun, G., Rozenberg, G., Salomaa, A.: Computing by splicing. Programmed and evolving splicing systems. In: IEEE Intern. Conf. on Evolutionary Computing, Indianapolis, pp. 273–277 (1997)
15. Păun, G., Rozenberg, G., Salomaa, A.: DNA Computing - New Computing Paradigms. Springer, Berlin (1998)
16. Pixton, D.: Regularity of splicing languages. Discrete Appl. Math. 69, 101–124 (1996)
17. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I-III. Springer, Berlin (1997)